IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appl. No.:	09/843,816
Conf. No.:	9954
Applicant:	Opsware Inc.
Filing Date:	April 30, 2001
Title:	Interface for Automated Deployment and Management of Network Devices
Examiner:	Benjamin A. Ailes
Art Unit:	2142
Docket No.:	PA3941US
Customer No.:	22830

DECLARATION OF JACOB McGUIRE UNDER 37 C.F.R. § 131

I, JACOB MCGUIRE, AM THE SOLE NAMED INVENTOR FOR THE ABOVE-REFERENCED PATENT APPLICATION AND HEREBY DECLARE AS FOLLOWS:

1. During the period of time the subject matter of this application was being developed, I was employed by Opsware Inc., previously named Loudcloud, Inc.

2. Prior to December 27, 2000, I conceived and reduced to practice the invention as claimed in this application.

3. Attached hereto as Exhibit 1 is code entitled Netdevlib.py, written on December 4, 2000. Netdevlib.py is a library containing generic commands that can be applied to network devices.

4. Attached hereto as Exhibit 2 is code entitled alteonlib.py, written on December 26, 2000. Alteonlib.py is a plug-in module that can register with a library (such as Netdevlib.py) and can operate to convert the generic commands from the library into device-specific commands and transmit the commands to remote individual devices of a type that are associated with alteonlib.py.

5. Attached hereto as Exhibit 3 is code entitled ciscopixlib.py, written on December 26, 2000. Ciscopixlib.py is another plug-in module that can register with a library (such as Netdevlib.py) and can operate to convert the generic commands from the library into a different device-specific commands (for a different device than the device-specific commands converted by the alteonlib.py module) and transmit the commands to remote individual devices of a type that are associated with ciscopixlib.py.

6. Attached hereto as Exhibit 4 is a directory listing identifying Exhibits 1, 2, and 3, as being created on December 4, 2000, December 26, 2000, and December 26, 2000 respectively.

7. I, therefore, claim priority under 37 C.F.R. § 131 over US nonprovisional application entitled "Automatic Configuration of a Data Storage System", published patent application number US 2002/0128815, filed on January 6, 2001 by Arif A. Merchant.

8. I hereby declare that all statements made herein of my own knowledge are true and that all statements made herein on information and belief are believed to be true and, further, that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001 and that such willful false statements may jeopardize the validity of this application or any patent issued thereon.

Date: May 1st, 2007

Jacob McGuire

Exhibit 1

Netdevlib.py

```
#
#  Netdevlib.py - a superclass for network devices
#

import string
import sys
import os
import re
import telnetpp2g
import socket
import time
from jiveutils import is_ip_address
from socket import gethostbyname
import types

class NetDev:
    def __init__(self, addr, tnobj = None, timeout=10):
        self.BadCmd = "BadCmd"
        if is_ip_address(addr):
            self.addr = addr
        else:
            self.addr = gethostbyname(addr)
        print "%s addr: %s" % (self.devtype, self.addr)
        if tnobj == None:
            self.conn = telnetpp2g.Telnet(addr, timeout)
        else:
            self.conn = tnobj
        self.log = "%s @ %s=n" % (self.devtype, addr)
        self.debuglevel = 0
        self.config_prompt = re.compile("\(.+\)#")
        self.enabled = 0

    #
    #  Given a device, not in enable mode, returns a dictionary that
    #  can be passed to spin.Device.updateHW
    #
    def truthify(self):
        pass

    #
    #  Given a new device, performs all authentication which needs to
    #  be done to get into the most privleged mode on that device
    #
    def enable_mode(self, passwords = None):
        pass

    #
    #  Given a device, not in enable mode, saves the running configuration
    #  into self.conf, and returns said configuration.
    #
    def get_config(self):
        pass

    #
    #  Given a device which has already had get_config() called on it, performs
    #  any processing on the configuration required before saving it into
    #  /cust/configs/<hostname>-config
```

```
    #
    def dump_config(self):
        pass

import alteonlib, ciscoioslib, ciscopixlib, ciscocatoslib, brocadelib,
netscreenlib

def conn_from_spin(spin, dvc_id):
    dvc = spin.Device.get({"id" : dvc_id})
    if dvc["dvc_type"] == "FIREWALL":
        dc = spin.Device.getChildren({"id" : dvc_id,
                                      "child_class" : "DeviceConsole"})[0]
        ip = dc["console_ip"]
        dvc_class = ciscopixlib.CiscoPIX
    elif dvc["dvc_type"] == "LOADBALANCER":
        ip = spin.Device.getIPList({"id" : dvc_id})[0]
        dvc_class = alteonlib.Alteon
    elif dvc["dvc_type"] == "ROUTER":
        ip = spin.Device.getIP({"id" : dvc_id})
        dvc_class = ciscoioslib.CiscoIOS
    elif dvc["dvc_type"] == "VPN":
        ip = spin.Device.getIP({"id" : dvc_id})
        dvc_class = netscreenlib.NetScreen
    elif dvc["dvc_type"] == "SWITCH":
        ip = spin.Device.getIP({"id" : dvc_id})
        if string.find(dvc["os_version"], "IOS") != -1:
            dvc_class = ciscoioslib.CiscoIOS
        elif dvc["dvc_mfg"] == "Brocade":
            dvc_class = brocadelib.Brocade
        else:
            dvc_class = ciscocatoslib.CiscoCatOS
    if type(ip) != types.StringType:
        print ip
    return dvc_class(ip)
```

Exhibit 2

Alteonlib.py

```
#! /lc/bin/python
#
#  Interact with a Cisco PIX firewall
#

import string
import sys
import os
import re
import telnetpp2g
import socket
import time
from netdevlib import NetDev
from jiveutils import swint

class Alteon(NetDev):
    devtype = "Alteon"
    def __init__(self, addr, tnobj = None, timeout=10):
        self.alteon_prompt_regex = re.compile(">> [\w|\d|\s]+# $")
        self.oldpw = None
        NetDev.__init__(self, addr, tnobj, timeout)

    def send_cmd(self, cmd, resp = None, wait = 5):
        if resp == None:
            resp = [self.alteon_prompt_regex]
        self.conn.write(cmd)
        idx, matchobj, response = self.conn.expect(resp, wait)
        self.log = self.log + response
        if (idx == -1):
            print "Error from", self.addr
            print "Loadbalancer failed to return to prompt in", repr(wait),
"seconds.  Barfing."
            print "Got", response
            print "Expected", resp[0].pattern
            self.log = self.log + "ERROR: last command: " + cmd
            raise self.BadCmd
        return idx, response

    def get_config(self):
        if self.enabled == 0:
            self.enable_mode()
        idx, self.conf = self.send_cmd("/cfg/dump\n")
        self.conn.close()
        return self.conf

    def connect(self):
        self.enable_mode()

    def dump_config(self):
        #
        #  Get rid of the /cfg/dump
        #
        self.conf = string.split(self.conf, "\n", 1)[1]
        self.conf = string.replace(self.conf, "\r", "")
        timestamp_regex = re.compile("\s*\d+:\d+:\d+ \w.+$", re.M)
        self.conf = timestamp_regex.sub("", self.conf)
```

```
        hostname_regex = re.compile('/cfg/snmp/name \"(\w+(-\w+)?\.\w+)\"',
re.M)
        matchobj = hostname_regex.search(self.conf)
        if matchobj == None:
            print "No hostname found on Alteon ", self.addr
            hostname = self.addr
        else:
            hostname = matchobj.group(1)

        fd = open("/cust/configs/%s-config" % (hostname), "w")
        fd.write(self.conf)
        fd.close()

        fd = open("/cust/configs/%s-log" % (hostname), "w")
        fd.write(self.log)
        fd.close()

    #
    #  config_mode() - this is just a placeholder to fill in the
    #                  network device signature until it is all
    #                  inherited properly
    #
    def config_mode(self):
        self.config_prompt = self.alteon_prompt_regex
        pass

    #
    #  enable_mode() - another placeholder
    #
    def enable_mode(self, passwords = None):
        password_re = re.compile("password:")
        yesno_prompt = re.compile("(\[y/n\]:)|(n to skip it)")
        idx, matchobj, response = self.conn.expect([password_re], 5)
        if (idx == -1):
            print response
            print "Unable to connect to the loadbalancer at ", self.addr, "."
            self.log = self.log + response
            raise self.BadCmd
        idx, resp = self.send_cmd("On3&R00t\n", [self.alteon_prompt_regex,
                                                 password_re,
                                                 yesno_prompt])
        if idx == 1:
            self.send_cmd("admin\n")
            self.oldpw = 1
        if idx == 2:
            self.send_cmd("y\n")
        self.send_cmd("lines 0\n")
        self.enabled = 1

    #
    #  get_arp_table
    #
    #  returns a list of tuples of the form (mac address, ip address)
    #
    def get_arp_table(self):
        self.enable_mode()
        idx, jive = self.send_cmd("/info/arp/dump\n")
```

```
        jive = string.split(jive, "\r\n")
        self.conn.close()
        arplines = jive[3:-2]
        alt_arp = []
        for arp in arplines:
            mac = arp[24:41]
            newmac = string.upper(mac)
            print newmac
            alt_arp.append((newmac, (string.strip(arp[2:17]))))
        return alt_arp

    def end_config(self):
        yesno_prompt = re.compile("\[y/n\]")
        self.send_cmd("apply\n")
        self.conn.write("save\n")
        self.conn.expect([yesno_prompt], 5)
        # this confirms the save
        self.send_cmd("y\n", [yesno_prompt])
        # this is to answer the switching boot config question
        self.send_cmd("y\n")
        self.send_cmd("/boot/reset\n", [yesno_prompt])
        self.conn.write("y\n")

    def done_config(self):
        self.end_config()

    def init_from_file(self, templatefile, vars):
        login_prompt = re.compile("Username:")
        pass_prompt = re.compile("Password:")
        lpass_prompt = re.compile("[Pp]assword:")
        yesno_prompt = re.compile("(\[y/n\]:)|(n to skip it)")

        #
        #  Authenticate past terminal server
        #

        idx, matchobj, response = self.conn.expect([login_prompt], 15)
        if (idx == -1):
            print response
            print 'No login prompt.'
            raise self.BadCmd

        print "sending username"
        self.send_cmd("jake\n", [pass_prompt])
        print "sending passwd"
        self.conn.write("qdSeT1E\n\r\n\r\n")
        time.sleep(2)
        self.conn.write("\r\n\r\n")
        good = None
        while 1:
            idx, matchobj, response = self.conn.expect([lpass_prompt], 5)
            print idx, response
            if idx == 0:
                good = 1
            else:
                break
        #
```

```
        #  if good is true, we should have gotten to the last yes/no prompt
        #
        print good
        if good == 1:
            self.conn.write("admin\n")
            self.conn.expect([yesno_prompt], 5)
            self.send_cmd("n\n")
        fd = open(templatefile, "r")
        config = fd.read()
        fd.close
        for var, val in vars.items():
            config = string.replace(config, var, val)
        cfglines = string.split(config, "\n")
        for cfgline in cfglines:
            # when the alteons are processing scripts, they don't
            # give back a prompt.  this is crude rate-limiting
            self.conn.write(cfgline + "\n")
            time.sleep(0.25)

        # you can't set the administrator password from a script

        self.send_cmd("\n")
        self.send_cmd("/cfg/sys/admpw\n", [lpass_prompt])
        self.send_cmd("admin\n", [lpass_prompt])
        self.send_cmd("On3&R00t\n", [lpass_prompt])
        self.send_cmd("On3&R00t\n")
        self.end_config()

    def truthify(self):
        self.get_config()

        model_re = re.compile(r"script start \"([\w\s]+)\"")
        vlan_re = re.compile(r"/cfg/vlan (\d+)/ena/name \"([\w\s]+)\"")
        if_re = re.compile(r"/cfg/ip/if (\d+)/addr ([\d.]+)/mask [\d.]+/broad
[\d.]+/ vlan (\d+)")
        version_re = re.compile("Version ([\d+.]+)")

        sysname = "ERROR"
        model = "ERROR"
        version = "ERROR"

        vlan_list = vlan_re.findall(self.conf)
        if_list = if_re.findall(self.conf)

        try:
            model = model_re.search(self.conf).group(1)
            sysname = string.split(string.split(self.conf, "\n")[4])[1]
            version = version_re.search(self.conf).group(1)
        except AttributeError, why:
            pass

        vlans = {}
        for number, name in vlan_list:
            vlans[number] = name

        ifs = []
        for slot, addr, vlan in if_list:
```

```
        int = swint(slot)
        int.type = "VLAN"
        int.ipaddr = addr
        if slot == "1":
            int.primary = "1"
        ifs.append(int.make_dict())

    sw_dict = {}
    sw_dict["mid"] = sysname
    sw_dict["dvc_model"] = model
    sw_dict["dvc_mfg"] = "Alteon"
    sw_dict["system_name"] = sysname
    sw_dict["os_version"] = version
    sw_dict["interface_cards"] = ifs
    sw_dict["dvc_type"] = "LOADBALANCER"

    return sw_dict
```

Exhibit 3

Ciscopixlib.py

```
#! /lc/bin/python
#
#  Interact with a Cisco PIX firewall
#

import string
import sys
import os
import re
import telnetpp2g
import socket
import time
import signal
from netdevlib import NetDev
from jiveutils import swint, convert_mac, is_ip_address

class CiscoPIX(NetDev):
    devtype = "CiscoPIX"
    def __init__(self, addr, tnobj = None, timeout=10):
        NetDev.__init__(self, addr, tnobj, timeout)

    #
    #  send_cmd - send a command to the firewall, and expect a certain
    #             response back
    #  cmd      - \n terminated string to send the firewall
    #  resp     - a list of regexs detailing permissable responses
    #
    #  returns (idx, response)
    #  idx      - index of the matching regex returned by the firewall
    #  response - text of the response from the firewall
    #
    #  raises BadCmd if the firewall does not respond with one of the
    #  permitted responses within 5 seconds
    #
    def send_cmd(self, cmd, resp, time = 10):
        self.conn.write(cmd)
        idx, matchobj, response = self.conn.expect(resp, time)
        self.log = self.log + response
        if self.debuglevel >= 1:
            print response
        if idx == -1:
            print "Switch output (", string.strip(response), \
                  ") did not match ", resp[0].pattern
            self.log = self.log + "ERROR: last command: " + cmd
            raise self.BadCmd, cmd
        else:
            return idx, response

    #
    #  get_config() - get the running config from the firewall
    #
    #  sets self.config to contain the text of the configuration
    #  of the firewall.  also returns the configuration.
    #
    def get_config(self):
        enable_prompt = re.compile("(\w+(\.\S+)?)#", re.M)
        switch_prompt = re.compile("(\w+(\.\S+)?)>", re.M)
```

```
    if self.enabled == 0:
        self.enable_mode()

    idx, self.conf = self.send_cmd("wr t\n", [enable_prompt])
    idx, self.ver = self.send_cmd("sh ver\n", [enable_prompt])
    self.send_cmd("disable\n", [switch_prompt])
    return self.conf
#
#  dump_config - write the running configuration into a file
#                <firewallname>-config in the current directory
#
def dump_config(self):
    self.conf = string.replace(self.conf, "\r", "")
    self.conf = string.split(self.conf, "\n", 2)[2]
    self.conf = string.split(self.conf, "[OK]")[0]
    hostname_re = re.compile("hostname (\w+(\.\S+)?)", re.M)
    matchobj = hostname_re.search(self.conf)
    self.hostname = matchobj.group(1)
    fd = open("/cust/configs/%s-config" % (self.hostname), "w")
    fd.write(self.conf)
    fd.close()
    fd = open("/cust/configs/%s-log" % (self.hostname), "w")
    fd.write(self.log)
    fd.close()

#
#  enable_mode() - connect to a firewall and bring it to enable mode
#
#  authenticates past the terminal server, turns of the pager (--More--)
#  and enables using the default password
#
def enable_mode(self, passwords = None):
    user_prompt = re.compile("Username:")
    lpass_prompt = re.compile("[P|p]assword:")
    switch_prompt = re.compile("(\w+(\.\S+)?)>", re.M)
    enable_prompt = re.compile("(\w+(\.\S+)?)#", re.M)

    idx, matchobj, response = self.conn.expect([user_prompt], 2)
    if (idx == -1):
        print 'No login prompt.'
        raise self.BadCmd
    idx, matchobj = self.send_cmd("jake\n", [lpass_prompt])
    if (idx == -1):
        print 'No password prompt'

    print "sending passwd"
    self.conn.write("qdSeT1E\n")
    self.conn.write("\r\n\r\n")
    time.sleep(2)
    self.conn.write("\r\n\r\n")
    good = None
    sent_uname = None
    sent_pass = None
    while 1:
        idx, matchobj, response = self.conn.expect([switch_prompt, \
                                                    enable_prompt, \
```

```
                                                    user_prompt, \
                                                    lpass_prompt], 2)
        if self.debuglevel >= 1:
            print response, ": ", idx
        if idx == 0:
            good = 1
        elif idx == 1:
            good = 2
        elif idx == 2:
            if sent_uname == None:
                self.conn.write("jake\n")
                sent_uname = 1
        elif idx == 3:
            if sent_pass == None:
                self.conn.write("qdSeT1E\n")
                sent_pass = 1
        else:
            break
    if good == 1:
        print "setting term len 0"
        print "sending enable"
        idx, respose = self.send_cmd("enable\n", [lpass_prompt, \
                                                  enable_prompt])
        if idx == 0:
            print "sending password"
            self.send_cmd("On3&R00t\n", [enable_prompt])
    elif good != 2:
        print "unable to enter enable mode"
        raise self.BadCmd, "Unable to enter enable made."

    print "Setting term length to zero"
    self.send_cmd("no pager\n", [enable_prompt])
    self.enabled = 1

#
#  config_mode - enter config mode from enable mode
#
def config_mode(self):
    self.config_prompt = re.compile("\(.+\)#")
    self.send_cmd("conf t\n", [self.config_prompt])

#
#  end_config - exit config mode and write the new configuration to memory
#
def done_config(self):
    enable_prompt = re.compile("(\w+(\.\S+)?)#", re.M)
    self.send_cmd("exit\n", [enable_prompt])
    self.send_cmd("wr mem\n", [enable_prompt], 10)

#
#  add_conduit() - add a conduit to a firewall
#  protocol      - either tcp or udp
#  address1      - local address/mask pair
#  port1         - local port (needs "eq", use "any")
#  address2      - remote address/mask pair
#  port2         - remote port (needs "eq")
#
```

```
#  note - can use "host a.b.c.d" instead of "a.b.c.d 255.255.255.255"
def add_conduit(self, protocol, address1, port1, address2, port2):
    config_prompt = re.compile("\(.+\)#")
    cmdstr = string.join(["conduit permit", "tcp", address1, \
                          port1, address2, port2, "\n"],. " ")
    self.send_cmd(cmdstr, [config_prompt])

#
#  ip_to_subnet() - given a host ip address and mask, returns the
#                   network address
#  ip             - ip address in dotted decimal format
#  mask           - subnet mask in dotted decimal format
#
#  returns network address in dotted decimal format
#
def ip_to_subnet(ip, mask):
    ip_octets = string.split(ip, ".")
    mask_octets = string.split(mask, ".")
    subnet_octets = []
    for i in range(4):
        subnet_octets.append(repr(string.atoi(ip_octets[i]) & \
                             string.atoi(mask_octets[i])))
    return string.join(subnet_octets, ".")

#
#  subnet_to_ip - returns the nth address in a subnet
#
#  ip           - network ip address
#  delta        - n (as in nth address)
#
#  returns the nth ip address in the subnet
def subnet_to_ip(self, ip, delta):
    subnet_octets = string.split(ip, ".")
    ip_octets = subnet_octets
    ip_octets[3] = repr(string.atoi(subnet_octets[3]) + delta)
    return string.join(ip_octets, ".")


#  init_from_file - initialize a firewall from a template file and a
#                   variable definition file
#
#  templatefile   - a file with a template config consisting of config
#                   statements with variables of the form $NAME
#  varfile        - a file with a list of $NAME <value> pairs, one per line
#
def init_from_file(self, templatefile, vars):
    user_prompt = re.compile("Username:")
    pass_prompt = re.compile("[P|p]assword:")
    new_fw_prompt = re.compile("pixfirewall>")
    new_fw_en_prompt = re.compile("pixfirewall#")
    enable_prompt = re.compile("(\w+(\.\S+)?)#", re.M)
    config_prompt = re.compile("\(.+\)#")

    #
    #  Authenticate past terminal server
    #
    idx, matchobj, response = self.conn.expect([user_prompt], 2)
```

```
if (idx == -1):
    print 'No login prompt.'
    raise self.BadCmd
idx, matchobj = self.send_cmd("jake\n", [pass_prompt])
if (idx == -1):
    print 'No password prompt'

print "sending passwd"
self.conn.write("qdSeT1E\n")
self.conn.write("\r\n\r\n")
time.sleep(2)
self.conn.write("\r\n\r\n")

#
#  Get to the pixfirewall> prompt
#
good = None
while 1:
    idx, matchobj, response = self.conn.expect([new_fw_prompt,
                                                new_fw_en_prompt], 2)
    print response
    if idx == 0:
        good = 1
    if idx == 1:
        good = 2
    else:
        break
if good == 1:
    self.send_cmd("enable\n", [pass_prompt])
    self.send_cmd("\n", [new_fw_en_prompt])
elif good == None:
    raise self.BadCmd

self.send_cmd("conf t\n", [config_prompt])
fd = open(templatefile, "r")
config = fd.read()
fd.close()
for var, val in vars.items():
    config = string.replace(config, var, val)
cfglines = string.split(config, "\n")
bannerend = None
for cfgline in cfglines:
    # we need nastiness in here to deal with the banner
    #
    # i should find a better way to handle this
    jive = string.split(cfgline)
    if len(jive) == 0:
        continue
    if cfgline[0] == "!":
        continue
    if jive[0] == bannerend:
        print "final banner line"
        biotch = biotch + cfgline + "\n"
        self.send_cmd(biotch + "\n", [config_prompt])
        bannerend = None
        continue
    if jive[0] == "banner":
```

```
                bannerend = jive[-1]
                print "Starting banner"
                biotch = cfgline + "\n"
                continue
            if bannerend != None:
                print "another banner line: ", cfgline
                biotch = biotch + cfgline + "\n"
                continue
            self.send_cmd(cfgline + "\n", [config_prompt])
        self.done_config()

    def truthify(self):
        self.get_config()

        sw_dict = {}

        sysname_re = re.compile("hostname (\S+)")
        version_re = re.compile("PIX Version (\S+)")
        serial_re = re.compile("Serial Number:\s+(\S+)")
        model_re = re.compile("Hardware:\s+(\S+),")
        mac_addr_re = re.compile("\d: (\w+): address is ((([0-9a-f]){4}.){2}([0-
9a-f]{4}))")
        ip_addr_re = re.compile("ip address (\S+) ([0-9\.]+) ([0-9\.]+)")
        int_names_re = re.compile("nameif (\S+) (\S+) security")

        try:
            model = model_re.search(self.ver).group(1)
            sysname = sysname_re.search(self.conf).group(1)
            serialnum = serial_re.search(self.ver).group(1)
            os_ver = version_re.search(self.conf).group(1)
        except AttributeError, why:
            pass

        sw_dict["mid"] = sysname
        sw_dict["dvc_model"] = model
        sw_dict["dvc_mfg"] = "cisco Systems"
        sw_dict["serial_num"] = serialnum
        sw_dict["system_name"] = sysname
        sw_dict["os_version"] = os_ver
        sw_dict["dvc_type"] = "FIREWALL"

        int_names = {}
        ints = {}

        ipaddrs = ip_addr_re.findall(self.conf)
        macaddrs = mac_addr_re.findall(self.ver)
        for slot, name in int_names_re.findall(self.conf):
            int_names[name] = slot

        for slot, mac, a, b, c in macaddrs:
            ints[slot] = swint(slot)
            ints[slot].macaddr = convert_mac(mac)
            if slot == "ethernet0":
                ints[slot].primary = "1"
            ints[slot].type = "ETHERNET"

        for name, ip, mask in ipaddrs:
```

```
            ints[int_names[name]].ipaddr = ip

        jive = string.split(sysname, ".")
        conname = string.join([jive[0]+"-con", jive[1]], ".")

        sw_dict["interface_cards"] = map(swint.make_dict, ints.values())
        sw_dict["device_console"] = [{"console_ip" : self.addr,
                                      "console_host_name" : conname}]

#        Wait for conduit support in spin
#        sw_dict["conduits"] = []
#        cfg_lines = string.split(self.conf, "\n")
#        for l in cfg_lines:
#            w = string.split(l)
#            if len(w) == 0 or w[0] != "conduit":
#                continue
#            sw_dict["conduits"].append(self.conduit_parser_spin(l))

        return sw_dict

    #
    #  Build a spin-compliant representation of a conduit
    #
    def conduit_parser_spin(self, l):
        conduit = {"conduit_desc":string.strip(l)}
        print l
        w = string.split(string.strip(l))
        if len(w) == 0 or not w[0] == "conduit":
            raise "NotConduit"
        #  Skip "conduit permit"
        pos = 2

        conduit["protocol"] = w[pos]
        pos = pos + 1

        #  we now are at the "local" address
        if w[pos] == "any":
            conduit["destination"] = "ANY"
            pos = pos + 1
            if w[pos] == "eq":
                conduit["port"] = w[pos+1]
                pos = pos + 2
        else:
            if w[pos] == "host":
                conduit["destination"] = w[pos+1]
                pos = pos + 2
            elif is_ip_address(w[pos]):
                conduit["destination"] = string.join(w[pos:pos+1])
                pos = pos + 2
            if w[pos] == "any":
                conduit["destination"] = "ANY"
                pos = pos + 1
            elif w[pos] == "eq":
                conduit["port"] = w[pos+1]
                pos = pos + 2

        #
```

```
    #  We are now at the remote address
    #
    if w[pos] == "any":
        conduit["source"] = "ANY"
        pos = pos + 1
    else:
        if w[pos] == "host":
            conduit["source"] = w[pos+1]
            pos = pos + 2
        elif is_ip_address(w[pos]):
            conduit["source"] = string.join(w[pos:pos+1])
            pos = pos + 2
        elif w[pos] == "any":
            conduit["source"] = "ANY"
            pos = pos + 1

    return conduit

#
#  conduit_parser(l)
#  Takes a configuration line representing a conduit and builds
#  a conduit data structure.
#
def conduit_parser(l):
    conduit = {}
    w = string.split(string.strip(l))
    pos = 1

    #
    #  should be permit or deny
    #  if spin, do not create
    #
    conduit["type"] = w[pos]
    pos = pos + 1

    #
    #  grab the protocol
    #
    conduit["protocol"] = w[pos]
    pos = pos + 1

    #
    #  we now are at the "local" address
    #
    if w[pos] == "any":
        conduit["local"] = {"objclass":"ANY"}
        pos = pos + 1
        if w[pos] == "eq":
            conduit["local"]["port"] = w[pos+1]
            pos = pos + 2
    else:
        f_dict = {}
        if w[pos] == "host":
            f_dict = {"objclass":"HOST"}
            f_dict["ip"] = w[pos+1]
            pos = pos + 2
        elif is_ip_address(w[pos]):
```

```
            f_dict = {"objclass":"NETWORK"}
            f_dict["subnet"] = w[pos]
            f_dict["mask"] = w[pos+1]
            pos = pos + 2
        if w[pos] == "any":
            f_dict["port"] = "ANY"
            pos = pos + 1
        elif w[pos] == "eq":
            f_dict["port"] = w[pos+1]
            pos = pos + 2
        conduit["local"] = f_dict

    #
    #  We are now at the remote address
    #
    if w[pos] == "any":
        conduit["remote"] = {"objclass":"ANY"}
        pos = pos + 1
    else:
        fdict = {}
        if w[pos] == "host":
            f_dict = {"objclass":"HOST"}
            f_dict["ip"] = w[pos+1]
            pos = pos + 2
        elif is_ip_address(w[pos]):
            f_dict = {"objclass":"NETWORK"}
            f_dict["subnet"] = w[pos]
            f_dict["mask"] = w[pos+1]
            pos = pos + 2
        if w[pos] == "any":
            f_dict["port"] = "ANY"
            pos = pos + 1
        elif w[pos] == "eq":
            f_dict["port"] = w[pos+1]
            pos = pos + 2
        elif w[pos] == "range":
            f_dict["port"] = w[pos+1] + "-" + w[pos+2]
            pos = pos + 3
        conduit["remote"] = f_dict

    return conduit

def list_conduits(self, spin):
    pass
```

Exhibit 4

Directory Listing

```
2000-11-07 08:17 jive/
2000-12-04 17:32 jive/base/
2000-10-22 14:03 jive/base/common/
2000-10-22 13:24 jive/base/common/footer.py
2000-11-02 14:44 jive/base/common/header.py
2000-10-22 13:24 jive/base/common/small_header.py
2000-10-22 14:03 jive/base/configs/
2000-10-22 14:03 jive/base/configs/__init__.py
2000-10-22 13:24 jive/base/configs/cfgblaster.py
2000-12-04 17:32 jive/base/configs/getallconfigs.py
2000-11-08 16:26 jive/base/configs/index.py
2000-11-29 16:45 jive/base/configs/list_devices.py
2000-10-22 13:24 jive/base/configs/map_switch.py
2000-11-08 16:26 jive/base/configs/oneconfig.py
2000-12-04 17:32 jive/base/configs/truthify.py
2000-10-22 14:03 jive/base/firewall/
2000-10-22 13:24 jive/base/firewall/add_conduits.py
2000-12-04 17:32 jive/base/firewall/choose_op.py
2000-12-04 17:32 jive/base/firewall/index.py
2000-12-04 17:32 jive/base/firewall/list_conduits.py
2000-10-22 14:03 jive/base/images/
2000-10-22 13:24 jive/base/images/guwar-l.gif
2000-10-22 13:24 jive/base/images/guwar-r.gif
2000-10-22 13:24 jive/base/images/pimphat-l.gif
2000-10-22 13:24 jive/base/images/pimphat-r.gif
2000-10-22 13:53 jive/base/oplets/
2000-10-22 13:53 jive/base/oplets/firewall/
2000-10-22 14:03 jive/base/testdir/
2000-10-22 13:24 jive/base/testdir/index.py
2000-10-22 14:03 jive/base/__init__.py
2000-12-04 17:32 jive/base/index.py
2000-12-04 17:32 jive/base/init.py
2000-10-22 13:24 jive/base/probeop.py
2000-10-22 13:24 jive/base/secret.py
2000-12-04 17:32 jive/base/buildout/
2000-12-04 17:32 jive/base/buildout/build_templates.py
2000-12-04 17:32 jive/base/buildout/index.py
2000-12-04 17:32 jive/base/buildout/help.html
2000-11-02 14:44 jive/base/cserver/
2000-12-04 17:32 jive/base/cserver/index.py
2000-12-04 17:32 jive/base/cserver/probeop.py
2000-12-04 17:32 jive/base/passwords.py
2000-12-04 17:32 jive/base/acl/
2000-12-04 17:32 jive/base/acl/index.py
2000-12-04 17:32 jive/base/acl/list_acls.py
2000-12-04 17:32 jive/base/acl/update_acl.py
2000-10-22 14:03 jive/crypto/
2000-10-22 13:24 jive/crypto/admin-ca.crt
2000-10-22 13:24 jive/crypto/jive.srv
2000-10-22 13:24 jive/crypto/opsware-ca.crt
2000-10-22 14:03 jive/etc/
2000-10-22 13:24 jive/etc/jive
2000-10-22 13:24 jive/etc/jive.args
2000-10-22 14:03 jive/mk/
2000-10-22 13:24 jive/mk/findmk.py
2000-11-08 16:26 jive/templates/
2000-10-22 13:24 jive/templates/cserver.txt
```

```
2000-12-04 17:32 jive/templates/fw000.txt
2000-11-07 08:17 jive/templates/slba.txt
2000-11-07 08:17 jive/templates/slbb.txt
2000-12-04 17:32 jive/templates/switcha.txt
2000-12-04 17:32 jive/templates/switchb.txt
2000-11-08 16:26 jive/templates/vpn.txt
2000-10-22 13:24 jive/Makefile
2000-10-22 14:03 jive/__init__.py
2000-12-11 20:54 jive/jive-LC.notes
2000-10-22 16:09 jive/jive-LC.spec
2000-12-11 20:54 jive/jive-LC.version
2000-11-07 08:17 jive/crypt
2000-11-07 08:17 jive/cryptpw.py
2000-11-08 16:26 jivelib/
2000-10-22 14:03 jivelib/mk/
2000-10-22 13:24 jivelib/mk/findmk.py
2000-10-22 13:24 jivelib/Makefile
2000-10-22 13:24 jivelib/__init__.py
2000-12-26 16:03 jivelib/alteonlib.py
2000-11-07 08:17 jivelib/brocadelib.py
2000-12-26 16:03 jivelib/ciscocatoslib.py
2000-12-26 16:03 jivelib/ciscoioslib.py
2000-12-26 15:23 jivelib/ciscopixlib.py
2000-11-29 16:56 jivelib/jivelib-LC.notes
2000-10-22 16:12 jivelib/jivelib-LC.spec
2000-11-29 16:56 jivelib/jivelib-LC.version
2000-12-04 17:32 jivelib/jiveutils.py
2000-12-04 17:32 jivelib/netdevlib.py
2000-10-22 13:24 jivelib/telnetpp2g.py
2000-11-08 15:06 jivelib/passwords.py
2000-12-01 17:09 jivelib/netscreenlib.py
```